COMSC-210 Lecture Topic 15
Binary Trees

Reference
Childs Ch. 15
YouTube DFS

Binary Tree As Linked Structure
data structure: the "root" pointer
`bool empty() const {return root == 0;}`
the node:

```
template <class T>
struct Node
{
  T data;
  Node* left;
  Node* right;
};
```

Binary Tree Iteration
#1 level order (array-based tree only)
all recursive (array-based or linked):
  #2 "Preorder": vertex, left child, right-child (VLR)
  #3 "Inorder": left child, vertex, right-child (LVR)
  #4 "Postorder": left child, right-child, vertex (LRV)

```
template <class T>
void inorderOutput(const Node* node) const
{
  if (!node) return;
  inorderOutput(node->left);
  cout << node->data;
  inorderOutput(node->right);
}
```

Binary Tree Dynamic Memory Management
generic recursive copy: from the root, using preorder sequence
recursive clear: from the root, using postorder sequence
copy constructor needs to copy
destructor needs to clear
assignment operator needs to clear and copy

Binary Tree Basic Getter Functions
`bool empty() const;` // `true` if `root` is zero
`int size() const;`// track #of entries in "operator[]" and "deleteKey"
`int height() const;` // a new one!
  counts number of "levels" *PDF*

Binary Search Trees (BST) *PDF*
balance issues *view*
full trees: 1 3 7 15...
complete trees
balanced trees *PPT*
"horizontal ordering"
"inorder" iteration

BST `operator[ ]`
the easy way (see sample code)
  simple iteration from root, looking to match a key
  max #of operations = height of tree, O(log n)

the hard way: maintain balance, and maintain O(log n)
  AVL height balancing *PDF*
    balance factor = right-child height - left-child height
    ...maintain in {-1, 0, +1} *PDF*
  red-black height balancing

BST `deleteKey`
maintain "horizontal ordering"
"seek & swap" algorithm *PDF*

the hard way: maintain balance, and maintain O(log n)
  AVL height balancing
  red-black height balancing